IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION PAPERS

OF

DAVID JOHN BUTCHER

HEDLEY JAMES FRANCIS

STEPHEN JOHN HILL

VLADIMIR VASEKIN

AND

ANDREW CHRISTOPHER ROSE

FOR

NULL EXCEPTION HANDLING

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the field of data processing systems. More particularly, this invention relates to the handling of exceptions due to program instructions making reference to null values.

Description of the Prior Art

It is known within data processing systems to provide mechanisms that check for memory accesses being made to memory locations specified by a null variable. When such conditions arise, processing is diverted to a null exception handler and appropriate corrective action taken before processing is restarted. Typically, these mechanisms are provided within a virtual machine software implementation with the null detection taking place in software.

In the case of the Jazelle Java hardware acceleration mechanisms produced by ARM Limited of Cambridge, England, a hardware block is provided which decodes Java bytecodes and generates control signals for use in controlling processing logic to perform desired processing operations. Such Jazelle hardware effectively provides native execution of Java bytecodes and as part of this the hardware makes a check for null values associated with Java bytecodes and generates an appropriate null value exception when these occur. This null value exception is then handled in software.

The hardware Jazelle approach suffers from the disadvantage that it is difficult to map some stack based programming languages into register file based processors in an efficient manner. This renders it desirable to use approaches such as just in time (JIT) compilation and dynamic adaptive translation. With these techniques, the non-native code is translated into native code which is then stored within memory and then subsequently executed. Whilst the approach of converting the non-native code into translated code and storing this in memory can result in more rapid execution once the translation has been made, it suffers from the disadvantage that a considerable amount of memory storage space is needed to store the translated program instruction stream. Measures which can reduce this storage space requirement are strongly advantageous.

SUMMARY OF THE INVENTION

Viewed from one aspect the present invention provides apparatus for processing data comprising:

processing logic operable to perform data processing operations; and

an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions; wherein

said instruction decoder is responsive to a memory access instruction:

(i) to compare a base register value stored within a base register specified by a base register field of said memory access instruction with a predetermined null value; and

(ii) if said base register value matches said predetermined value, then to branch to execution of a null value exception handler.

The present technique recognises that the provision of program instructions to check for the occurrence of null values introduces a significant performance reduction and code size increase since it is typically required to be performed frequently within the translated code. Even with undesirably complex JIT compilers or dynamic adaptive translators, there is still a need for additional program instructions to perform the null checks despite the compiler or translator attempting to optimse these out of the case where possible. The present technique recognises these disadvantages and provides that memory access instructions are checked to determine if the base register value corresponds to a predetermined null value. Within the context of a system which is executing translated program instructions, the predetermined null value can readily be chosen as one which would not in the normal course of execution of that translated program occur. If a match is detected between a base register value and the null value, then a null value exception handler is invoked. Modifying the memory access instructions to perform, at least when operating within a mode which is executing a translated instruction stream, a null value check reduces or removes the need to insert additional program instructions to perform this null value check and so advantageously reduces the code size. The null checking is able to trap inappropriate null values which occur due to programming errors in the program code being

translated as well as due to programming errors in the compilation or translation code itself.

It will be appreciated that the null value exception handler may be provided and may operate in many different ways. In one mechanism, the execution of the translated program may simply be stopped by the null value exception handler and an error reported. However, in preferred embodiments the memory access instruction serves to store a return address pointing to a memory location storing a program instruction to be executed upon return from the null value exception handler such that execution may be restarted when the null value handling has been completed. This return address pointer may resume execution at that specific point or merely at a point related to that specific point, such as restarting at a point prior to or subsequent to the execution point where the null value exception arose in a way that avoids a repeat of the exception.

The flexibility of the present technique is increased in embodiments in which the null value exception handler is located at a memory address pointed to by a value stored within a programmable configuration register. This programmable configuration register may advantageously be provided as a coprocessor configuration register since it will infrequently need changing and accordingly need not occupy one of the register within the main register file, which are typically a limiting resource.

The null value exception handling mechanism discussed herein may advantageously be provided in combination with other exception handling mechanisms which are accessed via a common pointer value and accordingly preferred embodiments serve to access the null value exception handler by referencing an instruction stored at a memory address given by the programmable configuration register subject to a fixed offset associated with the occurrence of the null value match.

The null value exception handler may deal with the null value exception in a variety of different ways as previously mentioned. In particular, preferred embodiments of the null value exception handler may differentiate between

occurrences of null value exceptions which are the result of trying to emulate a non-native instruction which is seeking to make a null value reference due to a programming error in the emulated code as distinct from an error resulting from a bug within a compiler or translator used to form translated code.

It will be appreciated that the non-native program instructions could take a wide variety of different forms such as, for example, Java bytecodes, MSIL bytecodes, CIL bytecodes and .NET bytecodes. The non-native instructions may also be native program instructions of a different apparatus, such as a different processor core (e.g. seeking to emulate x86 instructions on a RISCARM core).

The pointer to the null value exception handler could point directly to the start of this exception handling code or alternatively may point to a jump instruction which then jumps to the start of that exception handling code.

It will be appreciated that the memory access instructions may, for example, be either load instructions or store instructions, either of which can give rise to null value exceptions.

Viewed from another aspect the present invention provides a method of processing data with an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions, said method comprising the steps of:

in response to said memory access instruction decoded by said instruction decoder controlling said processing logic:

(i) to compare a base register value stored within a base register specified by a base register field of said memory access instruction with a predetermined null value; and

(ii) if said base register value matches said predetermined value, then to branch to execution of a null value exception handler.

Viewed from a further aspect the present invention provides a computer program product including a computer program operable to control an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions, said computer program comprising:

a memory access instruction decodable by said instruction decoder to control said processing logic:

(i) to compare a base register value stored within a base register specified by a base register field of said memory access instruction with a predetermined null value; and

(ii) if said base register value matches said predetermined value, then to branch to execution of a null value exception handler.

Viewed from a further aspect the present invention provides a computer program product including a computer program operable to translate non-native program instructions to form native program instructions directly decodable by an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions, said native program instructions comprising:

a memory access instruction decodable by said instruction decoder to control said processing logic:

(i) to compare a base register value stored within a base register specified by a base register field of said memory access instruction with a predetermined null value; and

(ii) if said base register value matches said predetermined value, then to branch to execution of a null value exception handler.

The above, and other objects, features and advantages of this invention will be apparent from the following detailed description of illustrative embodiments which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 schematically illustrates a data processing system which can be used to execute non-native program instructions;

Figure 2 schematically illustrates a translation of non-native program instructions to native program instructions, with those native program instructions being checked for null value exceptions;

Figure 3 is a flow diagram schematically illustrating processing operations which are controlled by an instruction decoder upon receipt of a memory access instruction;

Figure 4 schematically represents an instruction encoding for memory access instructions; and

Figure 5 schematically illustrates the relationship between a non-native program, a compiler translating that non-native program, a translated native program and the execution of that program.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 shows a data processing system 2 including a memory 4, a processor 6 and a configuration coprocessor 8. The memory 4 stores program instructions to be executed and data to be manipulated. The processor 6 includes processing logic including a register file 10, a multiplier 12, a shifter 14 and an adder 16. It will be appreciated by those in this technical field that the processor core 6 will typically contain many further circuit elements but these have been omitted for the sake of clarity.

An instruction decoder 18 serves to decode program instructions within a decode stage of an instruction pipeline 20 and to generate control signals which control the processing logic of the processor 6 to perform the desired processing operations. The configuration coprocessor 18 includes a programmable configuration register 22 storing a handler base pointer which serves to point to a memory location within the memory 4 which stores exception handling code, including null value exception handling code, or at least pointers thereto.

Figure 2 illustrates a stream of machine independent instructions BC1 to BC10 which are translated by a JIT compiler, or a dynamic adaptive translator, within a virtual machine to form a stream of native instructions which can be executed by the processor 6. As is illustrated, there may be a one-to-one, a one-to-many or a many-to-one mapping relationship. Some of the machine independent instructions are translated into memory access instructions, LDRs or STRs in this example assuming an ARM processor 6. These memory access instructions can potentially give rise to null value exceptions if they include a base register value a base register field pointing to a base register storing a null value. This null value may be "0" since it is often the case that normal emulated program execution will not legitimately access such a memory location (e.g. in the case of ARM processors that memory location is associated with hardware exception handling vectors and these would normally not be accessed by a program being emulated).

In the example of Figure 2, the first LDR instruction does give rise to a null value exception, as does the later STR instruction. The second LDR instruction does not give rise to a null value exception. When a null value exception occurs, a branch is made to a memory location specified by an address held within the configuration coprocessor register subject to a fixed offset, in this case "-4". In this example, the instruction at that location is a jump instruction which directs processing to the start of the null exception handler program proper, which is stored elsewhere. When the null exception handler program has completed, a return is illustrated as being made to an instruction following that which gave rise to the null value exception. It will be appreciated that in many cases the return could be made to a different location, or indeed no return made at all. When the null value exception is detected, a link register within the register file 10 is loaded with a value corresponding to a potential return location within the native instruction stream. This return value may or may not be used as mentioned above.

Figure 3 is a flow diagram illustrating the processing operations following a memory access instruction, at least some of which are controlled by the instruction decoder 18. At step 24, the instruction decoder 18 waits for receipt of a memory access

instruction. (It will be appreciated that the instruction decoder 18 is also simultaneously responsive to native program instructions of many different other types.) When a memory access instruction is received, processing proceeds to step 26 at which the value stored within the base register is read. If this value, when subject to a comparison with a predetermined null value at step 28, results in a non-match with the null value, then the normal memory access instruction is executed at step 30 and the processing illustrated in Figure 3 terminates.

However, if the processing at step 28 results in a match with the null value, then a null value exception needs to be initiated. At step 32 a return address is stored within a register R14 (although this may not be used as previously mentioned depending upon the nature of the null value exception handling employed). At step 34, the program counter register R15 within the register file 10 is loaded with a value obtained by reading the configuration coprocessor register 22 and subtracting a fixed offset of four from this. At step 36, a branch is then made either directly or indirectly to the null value exception handler utilising the program counter value loaded at step 34.

At step 38 the null value exception handler is run (the exception handling code is not normally considered as providing processing operations controlled by the instruction decoder 18 directly as a consequence of the memory access instruction currently being decoded). If step 38 detects a virtual machine error as having given rise to the null value exception, then step 40 serves to correct that error before the processing is resumed. If a virtual machine error is not detected, then step 42 serves to correct the null value access which has occurred due to incorrect programming within the non-native program being emulated. At step 44, processing of the emulated program is resumed at a position dependent upon (but not necessarily directly given by) the link register value starred in register R14 of the register file 10.

Figure 4 schematically illustrates a possible encoding of load and store memory access instructions which may be subject to the null value checking operation described above.

Figure 5 illustrates a Java bytecode program 46 which is translated by a JIT compiler 48 within a Java virtual machine 50 to form a translated native program 52 in the form of Thumb-2 native program instructions. These native program instructions include memory access instructions which are subject to the null value exception checking mechanisms described above. It will be appreciated that a native program 52 which contains these instructions which are subject to the null value checking, as well as a compiler or translator 48 which generates such instructions are all considered as aspects of the current technique. The native instruction program 52 is subsequently executed by the processor 6 to emulate the desired operation of the Java bytecode program 46. The non-native instructions may alternatively be MSIL bytecodes, CIL bytecodes or .NET bytecodes.

Although illustrative embodiments of the invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes and modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims.